Technical Cybersecurity

GDB Command Details

Viewing and Using Registers

Viewing Registers

- *info reg*: View key registers and values
- *info all-reg*: View ALL registers
- *info reg ${register name}*: View a single register
- *p/x ${register name}*: Print the register value, assign to var

Writing to registers

- This can change control flow! (demo later)
- *set ${register name} = {value}*
- *set ${register name} {operation}= {amount}*
  - *+=, -=*

Memory

Examining Memory

- *x/nfu {address}*
  - n: a number of the associated unit to show
  - f: The format; s (string), i (instruction) x (hex)
  - u: The unit; b (byte), h (half word), w (word), g (giant)
- x/20xw $rbp
  - Shows 20 words in hex from the address in rbp
- x/20xg 0x7fffffffdce0
  - Shows 20 double words in hex from 0x7fffffffdce0

Can set memory too

- set {int} $rbp = 0x01
- set {int} 0x7fffffffdce0 = 0x01

Register Use

64-bit systems have 16 registers

- rax, rbx, rcx, rdx, rdi, rsi, rbp, rsp, r8-r15
  - rax: accumulator, stores return values from ops and functions
  - rbx: base register
  - rcx: count register; used with loops
  - rdx: data register; works with accumulator for computation
- rax, rcx, rdx, rdi, rsi, rsp, r8-r11: not saved across functions
- rbx, rbp, r12-r15: **are** saved across functions
- rdi, rsi, rdx, rcx, r8, r9: pass first six function params (larger use stack)

Register Layouts

Bytes 1-8	5-8	7-8	8
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rsp	esp	sp	spl
rbp	ebp	bp	bpl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r14	r15d	r15w	r15b

Other Registers

Segment Registers

- CS, SS, DS, ES, FS, GS: 16-bit
- Point to program segments (code, stack, data, etc.)
- Not relevant in 64-bit systems

Flags register

- eflags: Interrupt flag, carry flag, negative flag, etc.

Lots more

- We’re not usually interested in them though

Special Registers

RBP

- Base pointer register: reference variables on stack
- Was more useful in 32-bit systems when function args passed on stack

RSP

- Stack pointer register: points to the top of the stack

RIP

- Instruction pointer
- Points to the current instruction

32 & 64-bit

32-BIT REGISTERS

- eax, ebx, ecx, edx, esi, edi, ebp, esp
- Function args were kept on stack, ebp much more important as a result
- Basically, less registers, replace r (for register) with e (for extended)

IoT has very little 64-bit, very little x86 today

Let’s talk about ARM and MIPS next.